Analysing Twitter during the Ukraine-Russia conflict

Filippo Banti, Quoc-Bao Luu, Christophe Broillet

May 30, 2022

1 Introduction and data source

The network was developed starting from a dataset found on *Kaggle*[1], an assortment of tweets concerning the conflict between Russia and Ukraine started at the end of February 2022. The original dataset was composed by 5'867'587 tweets spanning through 12 days: from February 24 2022 to March 8 2022, organized in one file per day.
Each tweet on the dataset came with 17 attributes: `userid` author identification number, `username` author username, `acctdesc` author profile bio, `location` author location (gathered from the profile), `following` number of accounts followed by the author, `followers` author followers number, `totaltweets` author's total number of tweets published, `usercreatedts` creation date of the author account, `tweetid` post identification number, `tweetcreatedts` publication date, `retweetcount` number of retweet, `text` tweet corpus, `hashtags` hashtags used inside the corpus, `language` (machine detected) language identifier, `coordinates` geotag coordinates used on tweet pubblication, `favorite_count` number of likes, `extractedts` tweet gather date. The complete code the project is available on GitHub[2].

2 Data loading and preprocessing

To handle this amount of tweets the powerful *pandas*[3] library has been used, making available plenty of useful functions to deal with the data-set. With this collection of tools, the very first step has been to select between the tweets and their attributes in order to keep only the relevant contents. In this way, through the `language` attribute only the posts wrote in English has been selected and stored with this 4 attributes: `userid`, `tweetid`, `text`, `hashtags`. Thereafter, thanks to the function `drop_duplicates()` duplicated tweets were successfully discarded and, using `sample()`, 10'000 posts from each day were randomly selected, putting together a representative dataset, way more practical and easy to analyze.

Once done with the early processing, it was essential to find a way to enrich our data, to get the connections between users useful to create the network. To deal with the task the library *tweepy*[4] was pleasantly used to access the Twitter API and gather more "attributes" related to each tweet. Going a little bit in specific terms, the associated `tweetid` attribute was submitted for each tweets and, with this, it has been retrieved:

- **The original `tweetid`** in case the former tweet was a retweet.
- **Likers and retweeters** list of users that liked/retweeted the post.

To organize this new features two columns (i.e. attributes) were added to the data-frame:

- `author_id` nonetheless than the original *author_id* of the tweet.
- `original_tweet_id` representing the former id of the retweeted tweet.

Now, with these new elements it was finally possible to create an edge-lists useful for the networks generation. Once again, using the API access, three different edgelists were created: one for the retweets, containing the fields `retweeter_id` and `author_id`, one for the likes containing `liker_id` and `author_id`, and the last as the merge of the two first ones. There are then three different networks.

[1] https://www.kaggle.com/datasets/bwandowando/ukraine-russian-crisis-twitter-dataset-1-2-m-rows
[2] https://github.com/ChristopheBroillet/twitter-ukraine-conflict
[3] Pandas: https://pandas.pydata.org/
[4] Tweepy: https://www.tweepy.org/

3 Network analytics

3.1 Louvain method implementation

The main part that will be implemented by hand is the Louvain method[5]. It is an agglomerative (bottom-up) algorithm to detect communities inside networks, that is based on the modularity measure of the network. The algorithm moves nodes into communities such that the modularity of the network is maximized. The output of the Louvain method is a dendogram containing different levels of aggregated communities. This algorithm applies well on large networks such as social networks, as its complexity is linearithmic on the number of nodes $\mathcal{O}(n \log(n))$.

The main problems in the implementation were to avoid as much as possible the iterations, e.g. *for loops* or list comprehensions as well as to reduce the number of computations while iterating. To solve these problems, *dictionaries* were used as main data-structures. This allowed to (i) pre-compute features and variables and update them at runtime while (ii) getting direct access to them with a custom index. In this implementation, *dictionaries* were used to store for example the communities (label of the community as *key* and a set of nodes as *value*), or the corresponding community for a given node (node as *key* and label of the community as *value*). The computation of the modularity was also problematic, as it is one of the most called computation. To get the algorithm works as fast as possible, the code was split between the first passage and the other ones. In that way, the first passage takes some shortcuts. For example, in the implementation for the modularity the intersection is computed between the neighbors and the community of a node for the first passage instead of iterating over the sets for the other passages. This is possible because at the first passage, each edge has a weight of 1.

To catch the lines that were bottlenecks in the code, the `cProfile`[6] library was used. It displays the time and the number of calls for all different functions and methods during the run of the program. This library allowed to optimize various parts of the code, and to really see which functions needed to be refactored in a more efficient way.

Number of nodes	`NetworkX`	Our implementation
20110	0.339	0.727
194402	4.469	7.873
438753	13.239	26.811
1054720	45.357	82.225

Table 1: Runtime in seconds comparison **for the first passage**

This implementation is now compared to `louvain_partitions()` from `NetworkX` in terms of runtime and resulting number of communities for 4 different networks taken randomly by sampling the dataset, with different number of nodes. Table 1 compares the runtime of this implementation of the Louvain method and the implementation from `NetworkX` for the first passage. This algorithm runs about twice slower than the implementation from `NetworkX` for the four different networks. However it still conserves the linearithmic complexity. Table 2 shows the number of communities after 1, 2 or 3 passages of the Louvain method, comparing this implementation with `NetworkX`. After 1 passage both implementations output similar results. However they are quite different after passage 2 and 3. For two of the four networks, the implementation from `NetworkX` was not able to give a third passage. It could be that the presented implementation is not constructed the same way of the one from `NetworkX`, or the definition of passage is different between the two implementations, since `louvain_partitions()` "yields partitions for each level of the Louvain Community Detection Algorithm[7]".

3.2 Methodology and analysis

All pieces to perform the analysis of the networks are now present. The Louvain implementation presented in this project will be run on the three networks created. They are the following:

[5] https://arxiv.org/abs/0803.0476

[6] https://docs.python.org/3/library/profile.html#module-cProfile

[7] https://networkx.org/documentation/latest/reference/algorithms/generated/networkx.algorithms.community.louvain.louvain_partitions.html#networkx.algorithms.community.louvain.louvain_partitions

Number of nodes	NX, 1-P	Our, 1-P	NX, 2-P	Our, 2-P	NX, 3-P	Our, 3-P
20110	25	25	22	17	-	15
194492	73	71	51	30	50	19
438753	138	137	68	34	66	28
1054720	210	205	62	23	-	18

Table 2: Number of communities comparison after n passages (n-P)

1. Likes only (2669418 nodes, 4245301 edges and 755 unique authors)
2. Retweets only (520842 nodes, 787529 edges and 682 unique authors)
3. Both likes and retweets (2790891 nodes, 4466776 edges and 786 unique authors)

This allows to compare (i) the results from the likes and retweets separated and (ii) if the results by merging likes and retweets in a single network are different or similar. The number of detected communities are shown in the table 3, for 2 passages of the handmade implementation.

Passage	Likes only	Retweets only	Both likes and retweets
1	742	660	770
2	130	166	122

Table 3: Number of detected communities from Louvain, for the three different networks

The numbers of communities after the first passage are expected, as they are more or less the number of unique authors. Thus, the Louvain algorithm aggregates communities by people that liked or retweeted the same original tweet. The communities after the second passage are then more interesting to analyze.

The network that contains both the likes and the retweets will now be analyzed. The same analysis can be made on the two other networks. After the second passage in table 3, it appears that four communities contain each more than 10% of the nodes. Then, five contain between 7% and 10%. The other ones are tiny in comparison, as shown in the figure 1.

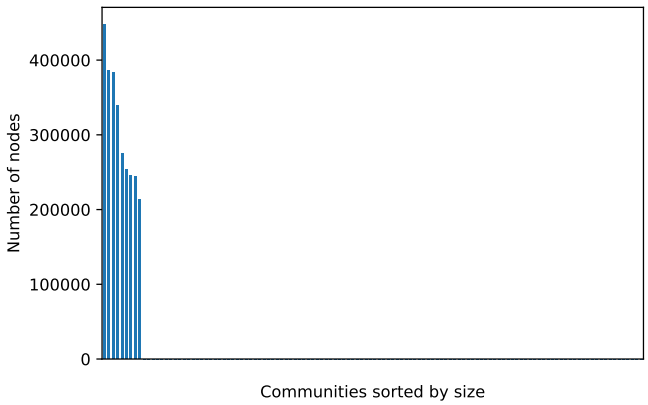

Figure 1: Number of nodes for each community for the likes and retweets graphs.

Then, some of the authors of the biggest community are extracted. It consists of namely *ZuzanaCaputova*[8], president of the Slovakia, and *Ukrayina1*[9]. The two authors were active in Twitter during the conflict, and were posting some tweets. The reason why they appear in the same community after the second passage of Louvain could be that they share the same audience. Indeed, both authors seem to be more pro-Ukraine than pro-Russia.

[8] https://twitter.com/ZuzanaCaputova
[9] https://twitter.com/Ukrayina1

	Likes	Retweets	Any
Likes	-	7.69%	27.77%
Retweets		-	8.69%
Any			-

Table 4: Intersection over union for authors in the biggest community

	Likes	Retweets	Any
Likes	-	0.00%	5.55%
Retweets		-	0.00%
Any			-

Table 5: Intersection over union for authors in the second biggest community

Finally the number of identical authors in the communities are compared using the three networks. Table 4 shows the comparison for the biggest communities and Table 5 for the second biggest one.

4 Network exploration

For the network exploration, the `Gephi`[10] software was used. Some statistics about the networks are shown in the table 6.

	Likes only	Retweets only	Both likes and retweets
Number of nodes	2669418	520842	2790891
Number of edges	4245301	787529	4466776
Average degree	1.59	1.51	1.60
Average clustering coefficient	0.405	0.048	0.405

Table 6: Some statistics for the three graphs

The average degree is about the same for the three networks. However, the average clustering coefficient is one order of magnitude lower for the retweets than the other graphs. The reason can be that there is much more triangles, i.e. 3 nodes that are connected to each other, in the likes than in the retweets. Indeed, people tend more to just like than retweet.

Figure 2 shows the degree distribution of the third network (likes and retweets). This distribution shows that the very majority (about 80%) of the nodes have a degree of 1, i.e. a simple user that liked or retweeted a tweet, while only a couple of nodes have a degree of more than 100'000.

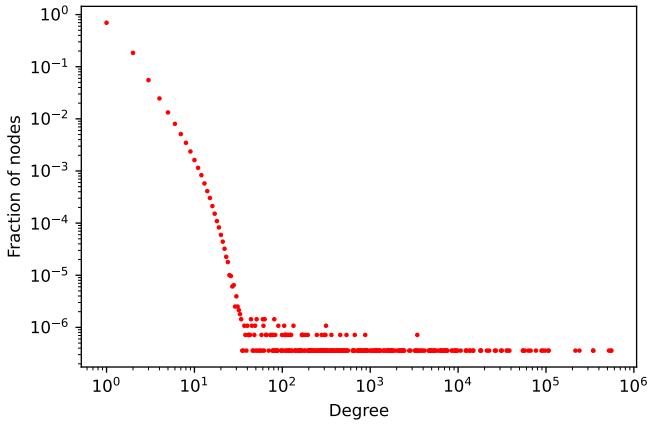

Figure 2: Degree distribution for the network containing both likes and retweets.

5 Network visualization

For the visualization, `Gephi` was used as well. This software created the graph from the edgelist for the network that contains both likes and retweets. The result is shown in figure 3.

[10] `https://gephi.org/`

Figure 3: Graph that represents both likes and retweets. Communities are colored.

6 Graph storage

As the graphs were stored in edgelists format after preprocessing the original data and calling the Twitter API, the graph storage part was decided not to be implemented, as it will not bring something useful additional.

7 Data enrichment

In this project, the data enrichment was not implemented. However, data could be enriched by taking new days of the original dataset and do the same process to get recent tweets.

8 Conclusion

The community detection of a Twitter network during the Ukraine-Russia conflict has not surprising results. Firstly, the number of communities are (after the first passage) more or less the number of unique authors of the networks. Then, the sizes of these communities are characteristic of social networks. A couple of communities consist of large parts of the whole network, while a high number of the communities contains only a tiny subset of the network. This leads to expected results, such as the clustering coefficient, the average degree of the network or the degree distribution.

The quality of the data can limit the results of this project. Indeed, tweets could carry misinformation or rumors that spread in the network, resulting in massive likes and retweets. Also, fake accounts that have been deleted in the meantime could also appear in this dataset. A preprocessing of the quality, such as taking only verified accounts, could lead to better and more meaningful results.

To push the project further, other analyses can be done. For example, misinformation diffusion can help to avoid one of the problem mentioned before. Additionally, to extract some other interesting results, sentiment analysis over users that posted the tweets can also be performed. This will show the different feelings of the authors in this dataset, as well as their likers and retweeters.